Chapter 1
Network Abstract Linear Programming with Application to Cooperative Target Localization*

Giuseppe Notarstefano and Francesco Bullo

Abstract We identify a novel class of distributed optimization problems, namely a networked version of abstract linear programming. For such problems we propose distributed algorithms for networks with various connectivity and/or memory constraints. Finally, we show how a suitable target localization problem can be tackled through appropriate linear programs.

1.1 Introduction

This paper focuses on a class of distributed computing problems and on its applications to cooperative target localization in sensor networks. To do so, we study abstract linear programming, that is, a generalized version of linear programming that was introduced by Matoušek, Sharir and Welzl in [1] and extended by Gärtner in [2]. Abstract linear programming is applicable also to some geometric optimization problems, such as the minimum enclosing ball, the minimum enclosing stripe and the minimum enclosing annulus. These geometric optimization problems are relevant in the design of efficient robotic algorithms for minimum-time formation control problems as shown in [3].

Giuseppe Notarstefano
Department of Engineering, University of Lecce, Via per Monteroni, 73100 Lecce, Italy,e-mail: giuseppe.notarstefano@unile.it

Francesco Bullo
Center for Control, Dynamical Systems and Computation, University of California at Santa Barbara, Santa Barbara, CA 93106, USA, e-mail: bullo@engineering.ucsb.edu

* This material is based upon work supported in part by ARO MURI Award W911NF-05-1-0219 and ONR Award N00014-07-1-0721. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 224428 (CHAT Project).

Linear programming and its generalizations have received widespread attention in the literature. The following references are most relevant in our treatment. The earliest (deterministic) algorithm that solves a linear program in a fixed number of variables subject to n linear inequalities in time $O(n)$ is given in [4]. An efficient randomized incremental algorithm for linear programming is proposed in [1], where a linear program in d variables subject to n linear inequalities is solved in expected time $O(d^2n + e^{O(\sqrt{d \log d})})$; the expectation is taken over the internal randomizations executed by the algorithm. An elegant survey on randomized methods in linear programming is [5]; see also [6]. The survey [7], see also [8], discusses the application of abstract linear programming to a number of geometric optimization problems. Regarding parallel computation approaches to linear programming, we only note that linear programs with n linear inequalities can be solved [9] by n parallel processors in time $O((\log\log(n))^d)$. The approach in [9] and the ones in the references therein are, however, limited to parallel random-access machines (usually denoted PRAM), where a shared memory is readable and writable to all processors. In this paper, we focus on networks described by arbitrary graphs.

The problem of target localization has been widely investigated and the related literature is therefore quite rich. Recently, the interest in sensor networks and distributed computation has lightened the attention on this problem from this new perspective. A good reference for localization and tracking in sensor networks is [10]. Two approaches may be used to tackle target localization, a stochastic and a deterministic one. As regards the deterministic approach (the one we use in this paper), a set membership estimation technique was proposed in [11]. Recently, a sensor selection problem for target tracking was studied in [12]. References for target localization and tracking in sensor networks, even by use of a stochastic approach, may be found therein.

The contributions of this paper are three-fold. First, we identify a class of distributed optimization problems that appears to be novel and of intrinsic interest. Second, we propose a novel simple algorithmic methodology to solve these problems in networks with various connectivity and/or memory constraints. Specifically, we propose three algorithms, prove their correctness and establish halting conditions. Finally, we illustrate how these distributed computation problems are relevant for distributed target localization in sensor networks. Specifically, we cast the target localization problem in the problem of approximating the intersection of convex polytopes by using a small number of halfplanes (among all those defining the polytopes). We show that suitable linear programs running in parallel, in fact, solve the problem, so that the proposed distributed algorithms may be used.

The paper is organized as follows. Section 1.2 introduces abstract linear programs. Section 1.3 introduces network models. Section 1.4 contains the definition of network abstract linear programs and the proposed distributed algorithms. Section 1.5 shows the relevance of the proposed distributed computing algorithms in the context of cooperative target localization.

Notation

We let $\mathbb{N}$, $\mathbb{N}_0$, and $\mathbb{R}_+$ denote the natural numbers, the non-negative integer numbers, and the positive real numbers, respectively. For $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f \in O(g)$ if there exist $n_0 \in \mathbb{N}$ and $k \in \mathbb{R}_+$ such that $|f(n)| \leq k|g(n)|$ for all $n \geq n_0$.

1.2 Abstract linear programming

In this section we present an abstract framework that captures a wide class of optimization problems including linear programming and various geometric optimization problems. These problems are known as *abstract linear programs* (or *LP-type problems*). They can be considered a generalization of linear programming in the sense that they share some important properties. A comprehensive analysis of these problems may be found for example in [7].

1.2.1 Abstract framework

We consider optimization problems specified by a pair (H, ω), where H is a finite set, and $\omega : 2^H \to \Omega$ is a function with values in a linearly ordered set $(\Omega, \leq)$; we assume that Ω has a minimum value $-\infty$. The elements of H are called *constraints*, and for $G \subset H$, $\omega(G)$ is called the *value* of G. Intuitively, $\omega(G)$ is the smallest value attainable by a certain objective function while satisfying the constraints of G. An optimization problem of this sort is called *abstract linear program* if the following two axioms are satisfied:

(i) *Monotonicity*: if $F \subset G \subset H$, then $\omega(F) \leq \omega(G)$;
(ii) *Locality*: if $F \subset G \subset H$ with $-\infty < \omega(F) = \omega(G)$, then, for all $h \in H$,

$$\omega(G) < \omega(G \cup \{h\}) \implies w(F) < w(F \cup \{h\}).$$

A set $B \subset H$ is *minimal* if $\omega(B) > \omega(B')$ for all proper subsets B' of B. A minimal set B with $-\infty < \omega(B)$ is a *basis*. Given $G \subset H$, a *basis of* G is a minimal subset $B \subset G$, such that $-\infty < \omega(B) = \omega(G)$. A constraint h is said to be *violated* by G, if $\omega(G) < \omega(G \cup \{h\})$.

The *solution* of an abstract linear program (H, ω) is a minimal set $B_H \subset H$ with the property that $\omega(B_H) = \omega(H)$. The *combinatorial dimension* δ of (H, ω) is the maximum cardinality of any basis. Finally, an abstract linear program is called *basis regular* if, for any basis with $\text{card}(B) = \delta$ and any constraint $h \in H$, every basis of $B \cup \{h\}$ has the same cardinality of B. We now define two important primitive operations that are useful to solve abstract linear programs.

(i) *Violation test*: given a constraint h and a basis B, it tests whether h is violated by B; we denote this primitive by `Viol`(B, h);

(ii) *Basis computation*: given a constraint h and a basis B, it computes a basis of $B \cup \{h\}$; we denote this primitive by $\texttt{Basis}(B,h)$.

Remark 1 (Examples of abstract linear programs). We present three useful geometric examples; see Figure 1.1.

(i) *Smallest enclosing ball:* Given n points in $\mathbb{R}^d$, compute the center and radius of the ball of smallest volume containing all the points. This problem has combinatorial dimension $d+1$.
(ii) *Smallest enclosing stripe:* Given n points in $\mathbb{R}^2$ in generic positions, compute the center and the width of the stripe of smallest width containing all the points. This problem has combinatorial dimension 5.
(iii) *Smallest enclosing annulus:* Given n points in $\mathbb{R}^2$, compute the center and the two radiuses of the annulus of smallest area containing all the points. This problem has combinatorial dimension 4.

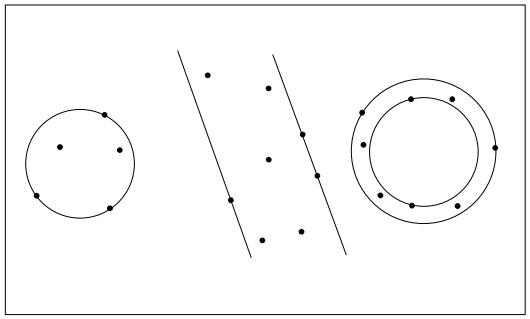

Fig. 1.1 Smallest enclosing ball, stripe and annulus

More examples are discussed in [1, 2, 5, 7]. □

1.2.2 Randomized sub-exponential algorithm

A randomized algorithm for solving abstract linear programs has been proposed in [1]. Such algorithm has linear expected running time in terms of the number of constraints, whenever the combinatorial dimension δ is fixed, and subexponential in δ. The algorithm, called $\texttt{SUBEX\_lp}$, has a recursive structure and is based on the two primitives introduced above, i.e., the violation test and the basis computation primitives. For simplicity, we assume here that such primitives may be implemented in constant time, independent of the number of constraints. Given a set of constraints G and a candidate basis $C \subset G$, the algorithm is as follows.

function SUBEX_lp(G,C)
if $G = C$, **then** return C
else
choose a random $h \in G \setminus C$
$B :=$ SUBEX_lp$(G \setminus \{h\}, C)$
if Viol(B,h), i.e., h is violated by B,
return SUBEX_lp$(G,$Basis$(B,h))$
else return B
end if
end if

For the abstract linear program (H,ω), the routine is invoked with SUBEX_lp(H,B), given any initial candidate basis B.

In [1] the expected completion time for the SUBEX_lp algorithm in conjunction with Clarkson's algorithms was shown to be in $O(d^2n + e^{O(\sqrt{d \log d})})$ for basis regular abstract linear programs. In [5] the result was extended to problems that are not basis regular.

1.3 Network models

Following [13], we define a synchronous network system as a "collection of *computing elements* located at nodes of a directed network graph." These computing elements are sometimes called *processors*.

1.3.1 Digraphs and connectivity

We let $I = \{1,\ldots,n\}$ and let $\mathscr{G} = (I,E)$ denote a directed graph, where I is the set of nodes and $E \subset I \times I$ is the set of edges. For each node i of $\mathscr{G}$, the number of edges going out from (coming into) node i is called *out-degree* (*in-degree*) and is denoted outdeg$[i]$ (indeg$[i]$). The set of outgoing (incoming) neighbors of node i are the set of nodes to (from) which there are edges from (to) i. They are denoted $\mathscr{N}_O(i)$ and $\mathscr{N}_I(i)$, respectively. A directed graph is called *strongly connected* if, for every pair of nodes $(i,j) \in I \times I$, there exists a path of directed edges that goes from i to j. In a strongly connected digraph, the minimum number of edges between node i and j is called the *distance from i to j* and is denoted dist(i,j). The maximum dist(i,j) taken over all pairs (i,j) is the *diameter* and is denoted diam$(\mathscr{G})$. Finally, we consider time-dependent directed graphs of the form $t \mapsto \mathscr{G}(t) = (I,E(t))$. The time-dependent directed graph $\mathscr{G}$ is *jointly strongly connected* if, for every $t \in \mathbb{N}_0$,

$$\cup_{\tau=t}^{+\infty} \mathscr{G}(\tau) \text{ is strongly connected.}$$

Moreover, the time-dependent directed graph $\mathcal{G}$ is *uniformly strongly connected* if, there exists $S > 0$ s.t. for every $t \in \mathbb{N}_0$

$$\cup_{\tau=t}^{t+S} \mathcal{G}(\tau) \text{ is strongly connected.}$$

1.3.2 Synchronous networks and distributed algorithms

Strictly speaking, a *synchronous network* is a directed graph $\mathcal{G} = (I, E_{\text{cmm}})$ where the set $I = \{1, \ldots, n\}$ is the set of *identifiers* of the computing elements, and the time-dependent map $E_{\text{cmm}} : \mathbb{N}_0 \to 2^{I \times I}$ is the *communication edge map* with the following property: an edge (i, j) belongs to $E_{\text{cmm}}(t)$ if and only if processor i can communicate to processor j at time t.

Definition 1 (Distributed algorithm). Let $\mathcal{G} = (I, E_{\text{cmm}})$ be a synchronous network. A distributed algorithm consists of the sets

- W, set of "logical" states $w^{[i]}$, for all $i \in I$;
- $W_0 \subset W$, subset of allowable initial values;
- M, message alphabet, including the `null` symbol;

and the maps

- $\text{msg} : W \times I \to M$, message-generation function;
- $\text{stf} : W \times M^n \to W$, state-transition function. □

Execution of the network begins with all processors in their start states and all channels empty. Then the processors repeatedly perform the following two actions. First, the ith processor sends to each of its outgoing neighbors in the communication graph a message (possibly the `null` message) computed by applying the message-generation function to the current value of $w^{[i]}$. After a negligible period of time, the ith processor computes the new value of its logical variables $w^{[i]}$ by applying the state-transition function to the current value of $w^{[i]}$, and to the incoming messages (present in each communication edge). The combination of the two actions is called a *communication round* or simply a round.

In this execution scheme we have assumed that each processor executes all the calculations in one round. If it is not possible to upper bound the execution-time of the algorithm, we may consider a slightly different network model that allows the state-transition function to be executed in multiple rounds. When this happens, the message is generated by using the logical state at the previous round.

The last aspect to consider is the *algorithm halting*, that is a situation such that the network (and therefore each processor) is in a idle mode. Such status can be used to indicate the achievement of a prescribed task. Formally we say that a distributed algorithm is in *halting status* if the logical state is a fixed point for the state-transition function (that becomes a self-loop) and no message (or equivalently the `null` message) is generated at each node.

1.4 Network abstract linear programming

In this section we define a *network abstract linear program* and propose novel distributed algorithms to solve it.

1.4.1 Problem statement

Informally we can say that a *network abstract linear program* consists of three main elements: a network, an abstract linear program and a mapping that associates to each constraint of the abstract linear program a node of the network. A more formal definition is the following.

Definition 2. A network abstract linear program (NALP) is a tuple $(\mathscr{G},(H,\omega),\mathscr{B})$ consisting of

(i) $\mathscr{G}=(I,E_{\text{cmm}})$, a communication digraph;
(ii) (H,ω), an abstract linear program;
(iii) $\mathscr{B}:H\to I$, a surjective map called *constraint distribution map*. □

The *solution* of the network abstract linear program is attained when all processors in the network have computed a solution to the abstract linear program.

Remark 2. Our definition allows for various versions of network abstract linear programs. Regarding the constraint distribution map, the most natural case to consider is when the constraint distribution map is bijective. In this case one constraint is assigned to each node. More complex distribution laws are also interesting depending on the computation power and memory of the processors in the network. In what follows, we assume $\mathscr{B}$ to be bijective. □

1.4.2 Distributed algorithms

Next we define three distributed algorithms that solve network abstract linear programs. First, we describe a synchronous version that is well suited for time-dependent networks whose nodes have bounded computation time and memory, but also bounded in-degree or equivalently arbitrary in-degree, but also arbitrary computation time and memory. Then we describe two variations that take into account the problem of dealing with arbitrary in-degree versus short computation time and small memory. The second version of the algorithm is suited for time-dependent networks that have arbitrary in-degree and bounded computation time, but are allowed to store arbitrarily large amount of information, in the sense that the number of stored messages may depend on the number of nodes of the network. The third algorithm considers the case of time-independent networks with arbitrary in-degree and bounded computation time and memory.

In the algorithms we consider a uniform network $\mathscr{S}$ with communication digraph $\mathscr{G} = (I, E_{\text{cmm}})$ and a network abstract linear program $(\mathscr{G}, (H, \omega), \mathscr{B})$. We assume $\mathscr{B}$ to be bijective, that is, the set of constraints H has dimension n, $H = \{h_1, \cdots, h_n\}$. The combinatorial dimension is δ.

Here is an informal description of what we shall refer to as the *FloodBasis* algorithm:

[Informal description] Each processor has a logical state of $\delta + 1$ variables taking values in H. The first δ components represent the current value of the basis to compute, while the last element is the constraint assigned to that node. At the start round the processor initializes every component of the basis to its constraint, then, at each communication round, performs the following tasks: (i) it acquires from its neighbors (a message consisting of) their current basis; (ii) it executes the SUBEX_lp algorithm over the constraint set given by the collection of its and its neighbors' basis and its constraint (that it maintains in memory), thus computing a new basis; (iii) it updates its logical state and message using the new basis obtained in (ii).

In the second scenario we work with a time-dependent network with no bounds on the in-degree of the nodes and on the memory size. In this setting the execution of the SUBEX_lp may exceed the communication round length. In order to deal with this problem, we slightly change the network model as described in Section 1.3, so that each processor may execute the state transition function "asynchronously", in the sense that the time-length of the execution may take multiple rounds. If that happens, the message generation function in each intermediate round is called using the logical state of the previous round. Here is an informal description of what we shall refer to as the *FloodBasisMultiRound* algorithm:

[Informal description] Each processor has the same message alphabet and logical state as in *FloodBasis* and also the same state initialization. At each communication round it performs the following tasks: i) it acquires the messages from its in-neighbors; ii) if the execution of the SUBEX_lp at the previous round was over it starts a new instance, otherwise it keeps executing the one in progress; iii) if the execution of the SUBEX_lp ends it updates the logical state and runs the message-generation function with the new state, otherwise it generates the same message as in the previous round.

In the third scenario we work with a time-independent network with no bounds on the in-degree of the nodes. We suppose that each processor has limited memory capacity, so that it can store at most D messages. The memory is dimensioned so to guarantee that the SUBEX_lp is always solvable during two communication rounds. The memory constraint is solved by processing only part of the incoming messages at each round and cycling in a suitable way in order to process all the messages in multiple rounds.

Here is an informal description of what we shall refer to as the *FloodBasisCycling* algorithm:

[Informal description] The first $\delta + 1$ components of the logical state are the same as in *FloodBasis* and are initialized in the same way. A further component is added. It is simply a counter variable that keeps trace of the current round. At each communication round each processor performs the following tasks: (i) it acquires from its neighbors (a message consisting of) their current basis; (ii) it chooses D messages according to a scheduled protocol, e.g., it labels its in-neighboring edges with natural numbers from 1 up to $\texttt{indeg}^{[i]}$

and cycles over them in increasing order; (iii) it executes the SUBEX_lp algorithm over the constraint set given by the collection of the D messages plus its basis and its constraint (that it maintains in memory), thus computing a new basis; (iv) it updates its logical state and message using the new basis obtained in (iii).

Remark 3. For the algorithm to converge it is important that each agent keeps in memory its constraint and thus implements the SUBEX_lp on the bases received from its neighbors together with its constraint. This requirement is important because of the following reason: no element of a basis B for a set $G \subset H$ needs to be an element in the basis of $G \cup \{h\}$ for any $h \in H \setminus G$. □

We are now ready to prove the algorithms' correctness.

Proposition 1 (Correctness of *FloodBasis*). *Let $\mathscr{S}$ be a synchronous time-dependent network with communication digraph $\mathscr{G} = (I, E_{\text{cmm}})$ and let $(\mathscr{G}, (H, \omega), \mathscr{B})$ be a network abstract linear program. If $\mathscr{G}$ is jointly strongly connected, then the* FloodBasis *algorithm solves $(\mathscr{G}, (H, \omega), \mathscr{B})$, that is, in a finite number of rounds each node acquires a copy of the solution of (H, ω), i.e., the basis B of H.*

Proof. In order to prove correctness of the algorithm, observe, first of all, that each law at every node converges in a finite number of steps. In fact, using axioms from abstract linear programming and finiteness of H, each sequence $\omega(B^{[i]}(t))$, $t \in \mathbb{N}_0$, is monotone nondecreasing, upper bounded and can assume a finite number of values. Then we proceed by contradiction to prove that all the laws converge to the same $\omega(B)$ and that it is exactly $\omega(B) = \omega(H)$. Suppose that for $t > t_0 > 0$ all the nodes have converged to their limit basis and that there exist at least two nodes, call them i and j, such that $\omega(B^{[i]}(t)) = \omega(B^{[i]}) \neq \omega(B^{[j]}) = \omega(B^{[j]}(t))$, for all $t \geq t_0$. For $t = t_0 + 1$, for every $k_1 \in \mathscr{N}_O(i)$, $B^{[i]}$ does not violate $B^{[k_1]}$, otherwise they would compute a new basis thus violating the assumption that they have converged. Using the same argument at $t = t_0 + 2$, for every $k_2 \in \mathscr{N}_O(k_1)$, $B^{[k_1]}$ does not violate $B^{[k_2]}$. Notice that this does not imply that $B^{[i]}$ does not violate $B^{[k_2]}$, but it implies that $\omega(B^{[i]}) \leq \omega(B^{[k_2]})$. Iterating this argument we can show that for every $S > 0$, every k connected to i in the graph $\cup_{t=t_0}^{t_0+S} \mathscr{G}(t)$ must have a basis $B^{[k]}$ such that $\omega(B^{[i]}) \leq \omega(B^{[k]})$. However, using the joint connectivity assumption, there exists $S_0 > 0$ such that $\cup_{t=t_0}^{t_0+S_0} \mathscr{G}(t)$ is strongly connected and therefore i is connected to j, thus showing that $\omega(B^{[i]}) \leq \omega(B^{[j]})$. Repeating the same argument by starting from node j we obtain that $\omega(B^{[j]}) \leq \omega(B^{[i]})$, that implies $\omega(B^{[i]}) = \omega(B^{[j]})$, thus giving the contradiction. Now, the basis at each node satisfies, by construction, the constraints of that node. Since the basis is the same for each node, it satisfies all the constraints, then $\omega(B) = \omega(H)$. ■

Remark 4. Correctness of the other two versions of the *FloodBasis* algorithm may be established along the same lines. For example, it is immediate to establish that the basis at each node reaches a constant value in finite time. It is easy to show that this constant value is the solution of the abstract linear program for the *FloodBasisMultiRound* algorithm. For the *FloodBasisCycling* algorithm we note that the procedure used to process the incoming data is equivalent to considering a time-dependent graph whose edges change with that law. □

Proposition 2 (Halting condition). *Consider a network $\mathscr{S}$ with time-independent, strongly connected digraph $\mathscr{G}$ where the* FloodBasis *algorithm is running. Each processor can halt the algorithm execution if the value of its basis has not changed after* $2\operatorname{diam}(\mathscr{G})+1$ *communication rounds.*

Proof. First, notice that, for all $t \in \mathbb{N}_0$ and for every $(i,j) \in E_{\text{cmm}}$,

$$\omega(B^{[i]}(t)) \leq \omega(B^{[j]}(t+1)). \tag{1.1}$$

This holds by simply noting that $B^{[j]}(t+1)$ is not violated by $B^{[i]}(t)$ by construction of the *FloodBasis* algorithm. Assume that node i satisfies $B^{[i]}(t) = B$ for all $t \in \{t_0, \ldots, t_0 + 2\operatorname{diam}(\mathscr{G})\}$, and pick any other node j. Without loss of generality assume that $t_0 = 0$. Because of equation (1.1), if $k_1 \in \mathscr{N}_O(i)$, then $\omega(B^{[k_1]}(1)) \geq \omega(B)$ and, recursively, if $k_2 \in \mathscr{N}_O(k_1)$, then $\omega(B^{[k_2]}(2)) \geq \omega(B^{[k_1]}(1)) \geq \omega(B)$. Iterating this argument $\operatorname{dist}(i,j)$ times, the node j satisfies $\omega(B^{[j]}(\operatorname{dist}(i,j))) \geq \omega(B)$. Now, consider the out-neighbors of node j. For every $k_3 \in \mathscr{N}_O(j)$, it must hold that $\omega(B^{[k_3]}(\operatorname{dist}(i,j)+1)) \geq \omega(B^{[j]}(t))$. Iterating this argument $\operatorname{dist}(j,i)$ times, the node i satisfies $\omega(B^{[i]}(\operatorname{dist}(i,j)+\operatorname{dist}(j,i))) \geq \omega(B^{[j]}(\operatorname{dist}(i,j)))$. In summary, because $\operatorname{dist}(i,j)+\operatorname{dist}(j,i) \leq 2\operatorname{diam}(\mathscr{G})$, we know that $B^{[i]}(\operatorname{dist}(i,j)+\operatorname{dist}(j,i)) = B$ and, in turn, that

$$\omega(B) \geq \omega(B^{[j]}(\operatorname{dist}(i,j))) \geq \omega(B).$$

This shows that, if basis i does not change for a duration $2\operatorname{diam}(\mathscr{G})+1$, then it will never change afterwards because all bases $B^{[j]}$, for $j \in \{1, \ldots, n\}$, have cost equal to $\omega(B)$ at least as early as time equal to $\operatorname{diam}(\mathscr{G})+1$. Therefore, node i can safely stop after a $2\operatorname{diam}(\mathscr{G})+1$ duration. ■

1.5 Distributed computation of the intersection of convex polytopes for target localization

In this section we discuss an application of network abstract linear programming to sensor networks, namely a distributed solution for target localization. We consider a set of (fixed) sensors $\{1,\dots,n\}$ deployed on a plane. These sensors have to detect a target located at position $x \in \mathbb{R}^2$. Each sensor i detects a region of the plane, $m^{[i]}(x) \subset \mathbb{R}^2$, containing the target; we assume that this region, possibly unbounded, can be written as the intersection of a finite number of half-planes. For $i \in \{1,\dots,n\}$, let c_i denote the number of half-planes defining the sensing region of sensor i. An example scenario with $c_i = 2$ for $i \in \{1,\dots,n\}$ is illustrated in Figure 1.2. From now on, in order to simplify the notation, we assume that $c_i = c$ for all $i \in \{1,\dots,n\}$, so that the number of half-planes (and thus the number of constraints) is nc.

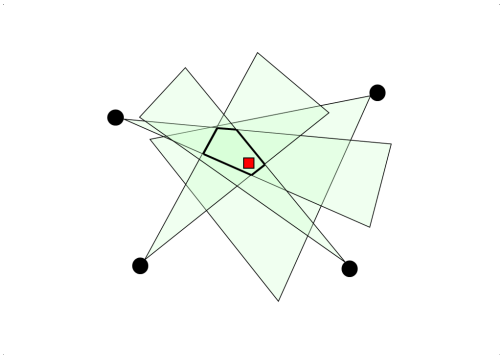

Fig. 1.2 Target localization: set measurements

The intersection of the regions detected by each sensor provides the best estimate of the target location, $M(x) = \cap_{i\in\{1,\dots,n\}} m^{[i]}(x)$. It easy to see that $M(x)$ is a non empty convex set, since it is the finite intersection of convex sets all containing the position x of the target.

Here, we are interested in approximating the intersection of convex polytopes by means of a "small" number of halfplanes. We consider the following approximation problem. Given a finite collection of convex polytopes with nonempty intersection, find the smallest axis-aligned rectangle that contains the intersection. We refer this rectangle as the "bounding rectangle."

The bounding rectangle has two important features. First, the rectangle provides bounds for the coordinates of the target. That is, let $p_o = (p_o^1, p_o^2)$ be the center of the rectangle with sides of length a and b respectively. For any $p = (p^1, p^2) \in M(x)$, then $|p^1 - p_o^1| \le a/2$ and $|p^2 - p_o^2| \le b/2$. Second, the bounding rectangle is characterized by at most four points of the polytope or, equivalently, by at most eight halfplanes.

It can be easily shown that computing the bounding rectangle is equivalent to solving four linear programs respectively in the positive and negative directions of each reference axis. More formally, let $v_\theta \in S^1$ be a vector forming an angle θ with the first reference axis. Given a set of half-planes $H = \{h_1, \ldots, h_n\}$, $h_i \subset \mathbb{R}^2$ for $i \in \{1, \ldots, n\}$, we denote (H, ω_θ) the linear program

$$\begin{aligned} &\min\ v_\theta^T x \\ &\text{subj. to } a_i^T x \leq b_i, \quad i \in \{1, \ldots, n\} \end{aligned}$$

where $h_i = \{x \in \mathbb{R}^2 \mid a_i^T x \leq b_i, a_i \in \mathbb{R}^2 \text{ and } b_i \in \mathbb{R}\}$. The bounding rectangle may be computed by solving the linear programs (H, ω_θ), $\theta \in \{0, \pi/2, \pi, 3\pi/2\}$. An example is depicted in Figure 1.3

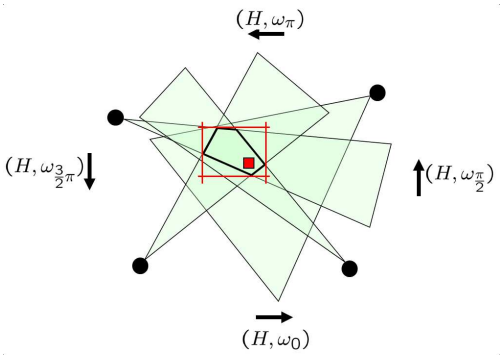

Fig. 1.3 Target localization: bounding rectangle

Remark 5. (i) A tighter approximation of the intersection may be obtained by choosing a finer grid for the angle θ. Choosing angles at distance $2\pi/k$, $k \geq 4$, the intersection is approximated by a k-polytope.

(ii) An inner approximation to a polytope in dimension d with n_p facets can also be computed via the largest ball contained in the polytope. This center and radius of this ball are referred to as the incenter and inradius of the polytope. It is known [14] that the incenter and the inradius may be computed by solving a linear program of dimension $d+1$ with n_p constraints.

(iii) The computation of a bounding k-polytope is a sensor selection problem in which one wants to select a few representative among a large set of sensors by optimizing some appropriate criterion. Indeed, the $2k$ sensors solving the problem are the only ones needed to localize the target. □

In the following we want to design a distributed algorithm running on a network to approximate the intersection of planar convex polytopes. We assume the sensor network may be described by the mathematical model introduced in Section 1.3. Let

$\mathscr{G} = (I, E_{\text{wsn}})$ be the associated communication graph, where the set $I = \{1, \ldots, n\}$ is the set of identifiers of the sensors and E_{wsn} is the communication edge map. We assume that $\mathscr{G}$ is a fixed undirected connected graph. We consider the network (abstract) linear programs defined by $(\mathscr{G}, (H, \omega_\theta), \mathscr{B})$ where (H, ω_θ) are the linear programs defined above and $\mathscr{B}$ is the mapping associating to each node the c constraints describing its sensing region. Therefore, the distributed algorithm to compute the bounding rectangle (k-polytope) consists of 4 (k) instances of *FloodBasis* running in parallel (one for each linear program). We denote *FloodRect* (*FloodPoly*) the algorithm consisting of the 4 (k) instances of *FloodBasis* running in parallel and the routine to compute the bounding rectangle (polytope).

Next, we are interested in estimating the position of a moving target. The proposed algorithm can be generalized according to the set membership approach, described for example in [11]. The idea is to track the target position by means of a prediction and "measurement update" iteration. The idea may be summarized as follows. We consider the sensor network described above, but with the objective of tracking a moving target. The sensors can measure the position of the target every $T \in \mathbb{N}$ communication rounds, so that during the T rounds they can perform a distributed computation in order to improve the estimate of the target. The target moves in the sensing area with bounded velocity $|v| \leq v_{\max}$. That is, given its position $x(t)$, the position after $\tau \in \mathbb{N}$ communication rounds may be bounded by $x(t+\tau) \in B(x(t), v_{\max}\tau)$, where we have assumed the inter-communication interval to be of unit duration.

In order to simplify notation we assume that each sensor can measure only one halfplane containing the target, i.e., we set $c = 1$. Also, we suppose that the sensors may keep in memory k past measures and use them to improve the estimate.

We begin the algorithm description by discussing the *prediction step*. Let $h^{[i]}(t) = \{x \in \mathbb{R}^2 \mid a^{[i]}(t)^T x \leq b^{[i]}(t)\}$ be the halfplane (containing the target) measured by sensor $s^{[i]}$ at time t. Since the target moves with bounded velocity, it follows easily that at instant $t+\tau$ the target will be contained in the halfplane $h^{[i]}(t+\tau|t) = \{x \in \mathbb{R}^2 \mid a^{[i]}(t)^T (x - v_{\max}\tau a^{[i]}(t)/\|a^{[i]}(t)\|) \leq b^{[i]}(t)\}$. The idea is illustrated in Figure 1.4.

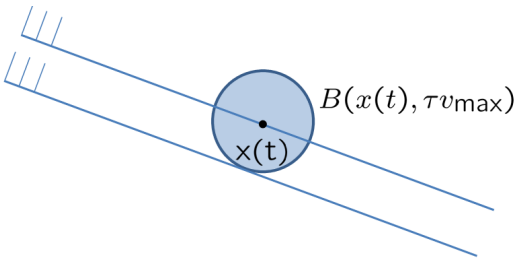

Fig. 1.4 Constraint after the prediction step

We are now ready to describe the estimation procedure. Informally, between two measurement instants each sensor runs a *FloodRect* algorithm such that each

FloodBasis instance has constraints given by the current measured halfplane and the prediction at time t of the latest k measures. More formally, each node solves the network linear programs $(\mathscr{G},(H(t),\omega_\theta),\mathscr{B})$, $\theta\in\{0,\pi/2,\pi,3\pi/2\}$, where $H(t)=\{H^{[1]}(t),\dots,H^{[n]}(t)\}$, with

$$H^{[i]}(t)=\{h^{[i]}(t|t-kT),h^{[i]}(t|t-(k-1)T),\dots,h^{[i]}(t|t-T),h^{[i]}(t)\}.$$

Then each node computes the corresponding bounding rectangle. The following result follows directly.

Proposition 3. *Let $t\in\mathbb{N}$ be a time instant when a new measure arrives. We denote $Rect^{[i]}(t+\tau)$ the estimate of the bounding rectangle at instant $t+\tau$ obtained by running the* FloodRect *algorithm for the network linear programs $(\mathscr{G},(H(t),\omega_\theta),\mathscr{B})$, $\theta\in\{0,\pi/2,\pi,3\pi/2\}$. Then*

(i) For any $\tau\in[0,T]$ and any $i\in\{1,\dots,n\}$, the rectangle $Rect^{[i]}(t+\tau)$ is a subset of the rectangle $Rect^{[i]}(t)$ and it contains the target.
(ii) For sufficiently large T, there exists $\tau_0\in[0,T]$ such that $Rect^{[i]}(t+\tau)=Rect^{[i]}(t+\tau_0)=Rect_0$ for all $\tau_0\le\tau\le T$, where $Rect_0$ solves the network linear programs.

Proof. To prove statement i) first note that for any $i\in\{1,\dots,n\}$ each constraints in $H^{[i]}(t)$ contains the target. Therefore each estimate of the bounding rectangle will contain the target. The monotonicity property of $\text{Rect}^{[i]}(t+\tau)$ follows by the monotonicity property of each *FloodBasis* algorithm solving the corresponding network linear program.

Statement ii) follows easily by the fact that each *FloodBasis* algorithm running in parallel solves the respective network linear program. ■

1.6 Conclusions

In this paper we have shown how to solve a class of optimization problems, namely abstract linear programs, over a network in a distributed way. We have proposed distributed algorithms to solve such problems in network with various connectivity and/or memory constraints. The proposed methodology has been used to compute an outer approximation of the intersection of convex polytopes in a distributed way. In particular, we have shown that a set approximation problem of this sort may be posed to perform cooperative target localization in sensor networks.

References

1. J. Matousek, M. Sharir, and E. Welzl, "A subexponential bound for linear programming," *Algorithmica*, vol. 16, no. 4/5, pp. 498–516, 1996.
2. B. Gärtner, "A subexponential algorithm for abstract optimization problems," *SIAM Journal on Computing*, vol. 24, no. 5, pp. 1018–1035, 1995.

3. G. Notarstefano and F. Bullo, "Network abstract linear programming with application to minimum-time formation control," in *IEEE Conf. on Decision and Control*, New Orleans, LA, Dec. 2007, pp. 927–932.
4. N. Megiddo, "Linear programming in linear time when the dimension is fixed," *Journal of the Association for Computing Machinery*, vol. 31, no. 1, pp. 114–127, 1984.
5. B. Gärtner and E. Welzl, "Linear programming - randomization and abstract frameworks," in *Symposium on Theoretical Aspects of Computer Science*, ser. Lecture Notes in Computer Science, vol. 1046, 1996, pp. 669–687.
6. M. Goldwasser, "A survey of linear programming in randomized subexponential time," *SIGACT News*, vol. 26, no. 2, pp. 96–104, 1995.
7. P. K. Agarwal and S. Sen, "Randomized algorithms for geometric optimization problems," in *Handbook of Randomization*, P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolim, Eds. Kluwer Academic Publishers, 2001.
8. P. K. Agarwal and M. Sharir, "Efficient algorithms for geometric optimization," *ACM Computing Surveys*, vol. 30, no. 4, pp. 412–458, 1998.
9. M. Ajtai and N. Megiddo, "A deterministic poly$(\log\log n)$-time n-processor algorithm for linear programming in fixed dimension," *SIAM Journal on Computing*, vol. 25, no. 6, pp. 1171–1195, 1996.
10. F. Zhao and L. Guibas, *Wireless Sensor Networks: An Information Processing Approach*. Morgan-Kaufmann, 2004.
11. A. Garruli and A. Vicino, "Set membership localization of mobile robots via angle measurements," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 450–463, 2001.
12. V. Isler and R. Bajcsy, "The sensor selection problem for bounded uncertainty sensing models," *IEEE Transactions on Automation Sciences and Engineering*, vol. 3, no. 4, pp. 372–381, 2006.
13. N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1997.
14. F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*, ser. Applied Mathematics Series. Princeton University Press, Sept. 2008, manuscript under contract. Available electronically at http://www.coordinationbook.info.